PROPOSAL FOR A COMAL 80 ERROR HANDLER

Prepared for the Second Standard Meeting, Copenhagen Februar 28. 1983

Edited by Lars Laursen, Horsens, Denmark

CBM COMAL GROUP FEBRUAR 28. 1983

Proposal for a COMAL 80 error handler

```
<structured statement> ::=
          <trap statement>

<trap statement> ::=

          TRAP <eol>
            <statement list>
          HANDLER <eol>
            <statement list>
          ENDTRAP <eol>

<numeric system function> ::=
          ERR !
          ERRFILE

<string system function> ::=
          ERRTEXT$

<simple statement> ::=
          <report statement>

<report statement> ::=
          REPORT !
          REPORT <error code>

<error code> ::=
          <numeric expression>
```

Evaluation of a trap-structure

The handler is divided in two blocks:

block 1: between TRAP and HANDLER:
Those statements, which have to be trapped.

block 2: between HANDLER and ENDTRAP:
Those statements, which have to be executed in case of an error in one of the statements between TRAP and HANDLER.

If no error arises in block 1, proceeds program execution after ENDTRAP.

If an error arises in block 1, program execution proceeds with block 2. The error number, the line number of the statement in which it arose, the error message, in the case of an file i/o error, also the file number, are remembered as long as block 2 is executed. I an error arise in block 2, execution proceeds in block 2 of an outer trap structure, if one is active, otherwise is the system handling the error (i.e. a message is printed). If a statement in block 1 is a procedure call or the statement contains a function call, and an error arises in one of the statements in the subprogram, it is regarded as if the error arises in the calling statement; i.e. the scope of names is the same in block 1 and block 2.

Example 1
=========

```
// program for calculation of factorial.

LOOP

  INPUT "Enter n: ": n

  fac:=1
  TRAP
    FOR i:=2 TO n DO
      fac:=fac*i
    ENDFOR i
    PRINT n,"! = ",fac
  HANDLER // assume overflow:
    PRINT "n too large."
  ENDTRAP

ENDLOOP
```

Example 2
=========

```
// function for calculation of factorial.

FUNC fac(n) CLOSED

  TRAP // catch 'overflow'
    RETURN fac'(n)
  HANDLER
    RETURN 1e+38 // largest number
  ENDTRAP

  FUNC fac'(n)
    IF n=0 THEN
      RETURN 1
    ELSE
      RETURN fac'(n-1)*n
    ENDIF
  ENDFUNC fac'

ENDFUNC fac
```

Report statement
================

REPORT without argument:
For use in block 2 of a trap structure. Lets the error, that caused execution of block 2, arise again (i.e. with the same line number as first time). This error may be caught by an outer trap structure or by the system. If the statement is used outside a block 2 of a trap structure, the error numbered 0 is issued ('no error to report').

REPORT with argument:
The argument is an integer in the range from 0 to 32767. The statement causes the error with the number given as argument. This error can be caught by an outer trap structure. If there is not an outer trap structure, the system will handle the error.

ERR function

This function has as value the number of that error, which caused the execution of the innermost block 2 of a trap structure. If there is no active block 2, it has the value 0.

ERRFILE function

This function has as value the number of that file, which was in use, when a file input/output error arose. If the error was not a file input/output error, or the function is called when there is no active block 2, then its value is 0.

ERRTEXT$ function

This function has as value a text, which describes the error, which caused the execution of block 2 of a trap structure. If there is no active block 2 , then its value is the null string.

Example 3

```
LOOP
  TRAP
    INPUT AT 10,1: "Enter a number: ": number
    EXIT // everything's all right
  HANDLER // error !!
    CASE ERR OF
    WHEN 206 // input error
      error("A number is expected")
    WHEN 2 // overflow
      error("The number is too large")
    OTHERWISE
      REPORT // give up
    ENDCASE
  ENDTRAP
ENDLOOP
```

Example 4

```
FUNC exists(filename$) CLOSED
  TRAP
    OPEN FILE 1,filename$,READ
    CLOSE FILE 1
    RETURN TRUE
  HANDLER
    CLOSE FILE 1
    RETURN FALSE
  ENDTRAP
ENDFUNC exists
```

Example 5
=========

```
parametererror:=300 // user defined error

FOR i:=-2 TO 90 DO
  TRAP
    PRINT USING "##! = ": i,
    PRINT fac(i)
  HANDLER
    IF ERR=parametererror THEN
      PRINT "undefined"
    ELSE
      REPORT
    ENDIF
  ENDTRAP
ENDFOR i

FUNC fac(n) CLOSED
  IMPORT parametererror
  IF n<0 THEN
    REPORT parametererror
  ELSE
    TRAP
      res:=1
      FOR i:=2 TO n DO res:=res*i
      RETURN res
    HANDLER // n too large
      RETURN 1e+38
    ENDTRAP
  ENDIF
ENDFUNC fac
```

Example 6
=========

```
FUNC getnum(row,col,prompt$) CLOSED
  DIM num$ OF 20
  LOOP
    INPUT AT row,col: prompt$+CHR$(22): num$
    TRAP
      RETURN VAL(num$) // return converted number
    HANDLER // catch all errors
      NULL // do nothing !!
    ENDTRAP
  ENDLOOP
ENDFUNC getnum

num1:=getnum(12,30,"Enter number 1: ")
num2:=getnum(13,30,"Enter number 2: ")
num3:=getnum(14,30,"Enter number 3: ")
```

Example 7
=========

```
PRINT CHR$(147), // clear screen
PRINT "Copy a sequential file."
PRINT
PRINT "Enter input- and output-file:"

DIM inpfile$ OF 20, outfile$ OF 20
inp:=2; out:=3

LOOP
  INPUT AT 5,3: "Input file : ": inpfile$
  INPUT AT 6,3: "Output file: ": outfile$
  PRINT AT 7,3: CHR$(22) // erase to end of line
  TRAP
    OPEN FILE inp,inpfile$,READ
    OPEN FILE out,outfile$,WRITE
    EXIT // everything's all right
  HANDLER
    CLOSE
    PRINT AT 7,3: reverse$(ERRTEXT$)
  ENDTRAP
ENDLOOP

// Copy the file:

TRAP
  WHILE NOT EOF(inp) DO
    PRINT FILE out: GET$(inp,5000), // copy 5Kb at a time.
  ENDWHILE
HANDLER
  IF ERRFILE=inp THEN
    ioerror(inpfile$)
  ELIF ERRFILE=out THEN
    ioerror(outfile$)
  ELSE
    REPORT
  ENDIF
ENDTRAP
CLOSE

PROC ioerror(filename$)
  PRINT AT 8,3: """"+filename$+""": "+reverse$(ERRTEXT$)
ENDPROC ioerror

FUNC reverse$(text$)
  RETURN CHR$(18)+" "+text$+" "+CHR$(18+128)
ENDFUNC reverse$

END "End of copy"
```

Error messages in CBM COMAL 80 rev. 2.00

Runtime errors, which can be trapped:

```
000:  no error to report
001:  function argument error
002:  overflow
003:  division by zero
004:  substring error
005:  value out of range
006:  step = 0
007:  dimension error
008:  error in print using
009:  illegal unit number
010:  index out of range
011:  name too long
```

Runtime errors, which cannot be trapped:

```
051:  system error
052:  stack overflow
054:  index/param error
055:  illegal number of indices
056:  string assignment error
057:  not implemented
058:  con not possible
059:  program has been modified
060:  verify error
061:  function value not returned
062:  not a variable
063:  program too big
064:  bad comal code
065:  not comal load file
066:  incompatible comal load file
067:  parameter lists are not equivalent
068:  no closed proc/func in file
069:  wrong number of parameters
070:  wrong index type
071:  ref parameter error
072:  wrong parameter type
073:  non-ram load
074:  checksum error in object file
075:  memory area is protected
076:  too many libraries
077:  not an object file
078:  no matching when
```

Syntax error messages:

```
100:  format error
101:  syntax error
102:  wrong type
103:  statement too long or too complicated
104:  statement only, not command
105:  array value parameter not implemented
106:  line number range: 1 to 9999
107:  too many names
108:  procedure/function does not exist
109:  structured statement not allowed here
110:  not a statement
111:  line numbers will exceed 9999
```

112: source protected!!!

Input/output error messages:
(can all be trapped)

200: end of data
201: end of file
202: file already open
203: file not open
204: not input file
205: not output file
206: input error
207: file input error
208: device not present
209: too many files open
210: read error (time out)
211: write error (time out)

212: short block on tape
213: long block on tape
214: checksum error on tape
215: end of tape
216: file not found

From 220-274 is disk error messages placed
(i.e. disk error number + 200)

220: read error (block header not found)
221: read error (no sync character)
222: read error (data block not present)
223: read error (checksum error in data block)
224: read error (byte decoding error)
225: write error (write-verify error)
226: write protect on
227: read error (checksum error in header)
228: write error (long data block)
229: disk id mismatch
230: syntax error (general syntax)
231: syntax error (invalid command)
232: syntax error (long line)
233: syntax error (invalid file name)
234: syntax error (no file given)
239: syntax error (invalid command)
250: record not present
251: overflow in record
252: file too large
260: write file open
261: file not open
262: file not found
263: file exists
264: file type mismatch
265: no block
266: illegal track and sector
267: illegal system t or s
270: no channel (available)
271: directory error
272: disk full
273: 4040: cbm dos v2 (dos mismatch)
8050: cbm dos v2.5
274: drive not ready

Dynamic syntax error messages:

<symbol> not expected
<symbol> missing
<symbol1> expected, not <symbol2>

Dynamic structure scan error messages:

<statement1> without <statement2>
<statement> missing
<statement1> expected, not <statement2>
<statement> not allowed in control structures
import allowed in closed proc/func only
wrong type of <statement>
wrong name in <statement>
<name>: name already defined
<name>: unknown label
illegal goto

Dynamic runtime error messages:
(cannot be trapped)

<name>: unknown statement or procedure
<name>: not a procedure
<name>: unknown variable
<name>: wrong type
<name>: wrong function type
<name>: not an array nor a function
<name>: not a variable
<name>: unknown array or function
<name>: wrong array type
<name>: import error
<name>: unknown package
<name>: array redefined
<name>: unknown label
<name>: name already defined
<name>: not a label
<name>: string not dimensioned
<name>: not a package